Trusted Geolocation-Aware Data Placement in Infrastructure Clouds

Nicolae Paladi
SICS Swedish ICT,
and Lund University, Sweden
nicolae@sics.se

Mudassar Aslam
SICS Swedish ICT,
and Mälardaren University, Sweden
mudassar@sics.se

Christian Gehrmann
SICS Swedish ICT,
Stockholm, Sweden
chrisg@sics.se

***Abstract*—Data geolocation in the cloud is becoming an increasingly pressing problem, aggravated by incompatible legislation in different jurisdictions and compliance requirements of data owners. In this work we present a mechanism allowing cloud users to control the geographical location of their data, stored or processed in plaintext on the premises of Infrastructure-as-a-Service cloud providers. We use trusted computing principles and remote attestation to establish platform state. We enable cloud users to confine plaintext data exclusively to the jurisdictions they specify, by sealing decryption keys used to obtain plaintext data to the combination of cloud host geolocation and platform state. We provide a detailed description of the implementation as well as performance measurements on an open source cloud infrastructure platform using commodity hardware.**

I. Introduction

Reliance on third-party providers for cloud storage and computing decouples data management from both data ownership and responsibility for correct data usage. A data owner loses control over the geographical placement of data once it is transferred to a cloud provider and earlier agreements become the only available tool to manage future data placement. In some cases, transfer of sensitive data to other countries is illegal and while agreements can be a basis for compensation, they can only help *post factum*, when the damage is already done.

The physical location of data storage and processing in cloud environments matters for several reasons: tax rates may differ based on where a transaction is conducted (rather than where the entity is registered); compliance rules or privacy laws may require that certain categories of data are not stored or processed in a different jurisdiction; finally, organizations with geographically distributed field offices might conduct operations – such as certain types research – which are illegal in some countries (e.g. stem cell research [1]). The above reasons are relevant to both data processing and storage. In [2], the authors clarify the importance of geolocation assurance mechanisms in cloud storage services. One of the central arguments is that data geolocation affects its confidentiality and privacy status. Similarly, [3]–[7] mention concerns – such as compliance requirements – about the geolocation of data in cloud computing environments in general and Infrastructure-as-a-Service (IaaS) in particular.

Calls for legal mitigation of issues related to data location in the cloud involve regulating cloud storage services through binding inter-government regulations [4]. However, this in only part of the solution, since agreements can be covertly breached – sometimes long before this fact is revealed to other parties.

Recent innovations in data center design – such as 'modular data centres' – improve the mobility of data centres. A central component of modular data centres are the standardized ISO 6346 weatherproof containers, capable of housing thousands of servers and necessary related components. This allows data center modules to be easily moved across large distances using standard transportation means. The idea was originally described in [8] and widely adopted since then. This approach differs from the traditional, static data centre architecture and along with advances in distributed storage systems architecture highlights the necessity to consider the geolocation of hosts when transferring data to a cloud storage provider.

Any practical solution for protection of data in cloud environments must consider its impact on functionality – a major driving force for adoption of cloud computing – such that e.g. distributed data processing capabilities are minimally affected. We propose a solution that combines *geolocation data* and *trusted computing* principles to allow data to be processed and transferred in plaintext only to geolocations approved by the data owner, without affecting data processing capabilities of distributed data stores. We present a prototype implementation based on Swift, a known distributed object storage system [9].

A. Contribution

Our contribution is as follows. First, we describe a protocol to securely store location information on cloud host platforms and later use this information to ensure that data is only available in plaintext on platforms that are placed in geolocations sanctioned by the data owner. Second, we provide a detailed implementation description of the above protocol, based on a popular cloud operating system and a known distributed object storage. Finally, we provide a security analysis of the chain of trust that allows to seal data to a given platform state extended to include the geolocation of the platform.

B. Organization

In § II we provide an overview of the related work that addresses data geolocation and data protection in cloud storage. We continue by defining the important terms in § III and presenting the system model in § IV. We present the protocol in § V, the detailed implementation in § VI and the security analysis in § VII. The prototype evaluation results are presented in § VIII, followed by a conclusion in § IX.

II. Background and related work

Related work on the topic has predominantly focused on establishing the fact that a certain piece of data *is* stored in a certain location, ignoring any potential replicas.

The problem of "data sovereignty", defined as "establishing data location at a granularity sufficient for placing it within the borders of a nation-state" was first introduced in [6]. The proposed solution combines provable data possession (PDP) schemes with a network-delay based protocol for data geolocation, in order to get a proof of the fact that the data is located in the respective data centre. This early paper lacks a specific adversary model and describes only a high-level solution.

In a follow-up, Gondree and Peterson propose a "constraints-based data geolocation" solution to determine the location of data and "bind" it to specific locations[1] [7]. The adversary model assumes an economically rational adversary aiming to reduce costs through data migration in spite of contractual agreements. The protocol assumes an initial model building stage, where landmarks ($\mathcal{L}$) throughout the analysed geographical region each build a latency-distance estimation model. Using this model, each landmark issues PDP challenges to the storage and generates a circular constraint of a radius centred on $\mathcal{L}$. The geolocation step of the protocol uses the intersection of geolocation constraints to determine the region where the data resides. The solution suffers from a series of limitations: it requires a set of landmarks close to the data centres of the cloud service provider; incorrectly assumes that the cloud service provider does not have dedicated communication channels between its data centres and finally, does not discuss location-based storage protection and rather just verifies that a certain file *is* placed on a given host.

The authors of [10] outline some ideas regarding the use of Trusted Platform Modules (TPM) on server platforms in the context of data location in cloud networks. The solution assumes that the identity of the server's TPM is stored along with the server's geographical position by the Certificate Authority and retrieved when needed. The solution further assumes a "Location verification and integrity check" module implemented in a hypervisor and suggests a two-phase protocol: the *initialization phase* includes remote attestation of the host and verification of its location; the *verification phase* includes a protocol to confirm the identity of the host based on communication with the TPM deployed on it. This solution is similar to our approach in the use of TPM as a hardware root of trust; however, it assumes that verification of the location is done through administrative methods, i.e. costly physical visit of the facilities. Furthermore, the paper does not describe any implementation results.

The National Institute of Standards and Technology (NIST) has described a proof of concept implementation for trusted geolocation in the cloud [5]. The proof of concept uses a combination of trusted computing, Intel Trusted Execution Technology (TXT) and a set of manual audit steps to verify and apply data location policies. The protocol establishes an automated hardware root of trust – defined as "inherently trusted combination of hardware and firmware that maintains the integrity of the geolocation information and the platform", in order to manage geolocation restrictions for hosts within an infrastructure cloud platform. The solution assumes that geolocation information is provisioned to the platform via an out-of-band mechanism and – along with platform metadata – stored in the TPM. This information is later accessed in order to verify the integrity of the host and the location of the platform. Similar to both our approach and [10], the use of TPM for platform identification offers a reliable, hardware-based root of trust. The solution in [5] assumes remote platform attestation – including location data – in order to establish the trustworthiness of the platform, which is a significant improvement compared to earlier work. However, we see several limitations of this approach and address them in this paper.

First, the protocol in [5] does not provide any cryptographic protection of data; rather, data placement is scheduled based on placement policies and thus data confidentiality depends on the correctness of the location policy. We believe this approach does not protect data from accidental or malicious policy misconfiguration, in which case plaintext data could be scheduled to an untrusted host. We address this by requiring that all uploaded client data is confidentiality and integrity protected and is only stored in plaintext *in the jurisdictions defined by the user*, a property achieved by performing remote attestation of the storage hosts and sealing the confidentiality and integrity protection keys to the platforms with a correct configuration.

Second, [5] assumes out-of-band provisioning of geolocation data to the storage hosts, without further clarification of the data format and delivery mechanisms. In this paper, we provide a detailed description of the format of data required for the geolocation of storage hosts in an infrastructure cloud. Furthermore, we address the question of secure out-of-band geolocation data delivery to storage hosts and also suggest a *complementary* geolocation acquisition model using dedicated GPS receivers.

Third, [5] does not describe a mechanism to re-provision geolocation tags and thus does not hold in the case of modular data centres mentioned in § I. Our proposed solution – which assumes a distributed geolocation information acquisition model – holds even in the cases when data hosts are relocated.

In [11] the authors discuss principles of domain-based storage protection in public infrastructure clouds. The principles outlined in the paper associate all objects stored in the IaaS cloud with explicit *storage domains*. A storage domain in this context corresponds to an organization or administrative unit that uses public cloud services (including the storage service) offered by the provider. All data in a single domain is protected with the same storage protection master key, the *domain key*. The paper further suggests that at guest VM launch, it is securely associated with a particular storage domain throughout its lifetime. Keys used for data encryption, decryption, integrity protection and verification in a single domain are derived by an external, trusted third party (TTP). We extend this protocol to include information about the geographical placement of data. We redefine the concept of "administrative domain" in [11] to also include a certain geographical area corresponding to a jurisdiction.

[1] Binding is here used in the sense of detecting occurrences of data misplacement, rather than data binding in the meaning common in trusted computing

Use of GPS signals in the context of data centres has been described in [12], where GPS and atomic clocks are used for time synchronization in order to implement externally-consistent distributed transactions.

Besides addressing the limitations of the above papers, our solution discusses cloud data storage protection including *replicas* of the data scattered throughout the distributed data store, something which – to the best of our knowledge – has not been done earlier.

III. Preliminaries

A. Definitions

IaaS (cloud) platform ($\mathcal{IP}$): We assume an IaaS platform model as defined by NIST in [13], which offers "processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications"; according to the same definition, users do not have control over the underlying infrastructure. IaaS platforms in this paper are assumed to include a large data store distributed over several data centres in distinct geographical areas.

User (U): Users are capable to access (read and write) data objects in the cloud data store. Let $U = \{u_1, \ldots, u_n\}$ be the set of all users of a certain $\mathcal{IP}$. Then, the set of all data objects that a certain user u_1 owns is denoted $f_1 = \{f_1^1, \ldots, f_n^1\}$.

Cloud Service Provider (CSP): We refer to a CSP as an entity that operates an $\mathcal{IP}$ and makes it available for *users*. The CSP includes both the case when the respective entity owns and physically manages its data centres and the case when an $\mathcal{IP}$ is deployed on computing resources provided by a third party supplier. The $\mathcal{IP}$ operated by the CSP may be deployed throughout arbitrarily many data centres.

Geolocation ($\mathcal{L}$): We refer to a *geolocation cell* $\mathcal{L}$ as a bounding area (e.g. country, region, territory, etc.) defined by a set of location points represented by their latitude and longitude ($l_i = lat_i, lon_i$) such that $\mathcal{L} = \{l_1, l_2, ...l_n\}$. Each l_i represents the location of an $\mathcal{IP}$ in the data centre; every data centre is associated with at most one $\mathcal{L}$ and no two geolocation cells overlap.

Jurisdiction ($\mathcal{J}$): We refer to a jurisdiction as "the territory or sphere of activity over which the legal authority of a court or other institution extends" [14]. Let $\mathcal{J}_i$, $\mathcal{J}_j$ be two jurisdictions with incompatible data protection regulations. Consider a user u_1 that operates on privacy-sensitive data and uses the services of a CSP with data centres present in both $\mathcal{J}_i$ and $\mathcal{J}_j$. For compliance reasons, u_1 may only process the data in $\mathcal{J}_i$ and faces penalties if data is processed or stored in plaintext in $\mathcal{J}_j$[2]. A valid jurisdiction is a non-empty set of $\mathcal{L}$s.

Trusted Platform Module (TPM): A TPM is a tamper-evident hardware cryptographic coprocessor built according to the specifications of the Trusted Computing Group [16]. In this work, we assume that all IaaS hosts underlying the $\mathcal{IP}$ are equipped with a TPM v1.2 chip. An active TPM records the software state of the platform at boot time and stores it in its platform configuration registers (PCRs) as a list of hashes. A TPM enables data protection by securely maintaining cryptographic keys, as well as through the set of functions it exposes. The *bind* and *seal* functions are particularly relevant for the proposed solution. According to [16], a message encrypted ("bound") using a particular TPM's public key is decryptable only by using the private key of the same TPM. Sealing is a special case of binding, where the encrypted messages produced through binding are only decryptable in a certain platform state (defined by the PCR values). This ensures that an encrypted message can only be decrypted by a platform found in a certain prescribed state. We refer to [16] for a detailed coverage of the bind and seal operations.

Trusted Third Party (TTP): The TTP is an entity which is trusted by the community and plays a key role in our protocol. The TTP is able to communicate with components deployed on compute hosts to exchange integrity attestation information, authentication tokens and cryptographic keys. In addition, the TTP can attest platform integrity based on the integrity attestation quotes provided by the TPM on the respective compute hosts, as well as seal data to a trusted configuration of the hosts. Finally, the TTP can verify the authenticity of a client as well as perform necessary cryptographic operations.

Trusted Platform (TP): In this paper, we define trusted platforms as server platforms the integrity and trusted state of which has been attested by the TTP. The trusted platforms of an $\mathcal{IP}$ comprise the *Trusted Computing Pool* ($\mathcal{T}$), introduced in [5], that is the collection of trusted platforms in a certain IaaS cloud platform.

B. Adversary model

We share the adversary model with [17], [18] which assume that privileged access rights can be maliciously used by CSP remote system administrators ($\mathcal{A}_r$). This scenario assumes that $\mathcal{A}_r$ can log in remotely to any host of the CSP and obtain root access. However, in this model $\mathcal{A}_r$ does not have physical access to the hosts. We add a geolocation aspect to the security model: u_1 requires assurance that her data is not stored or processed in plaintext outside jurisdiction $\mathcal{J}_i$. The CSP may experience intermittent errors and has an incentive to optimize costs by placing or processing data in a different jurisdiction, e.g. $\mathcal{J}_j$. We explicitly exclude Denial-of-Service attacks from our model, since we assume an economically rational CSP interested in maximizing its profits by continuing to provide services to users.

C. Problem Statement

Assume an authorized user u_1 writes a file f_1 to the storage provided by CSP. A trusted distributed storage system shall then satisfy the following properties:

1) The file f_1, as well as its replicas, must only be stored and processed in plaintext in the set of jurisdictions $\mathcal{J}_i$ defined by u_1.
2) The allowed jurisdictions $\mathcal{J}_i$ must be specified once, when f_1 is first written to the distributed storage. It shall be impossible for an adversary to subsequently change the association between f_1 and the set of allowed jurisdictions.

[2]Operating on encrypted text currently allows an impractically restricted set of operations [15]

3) Let f_1' be a file derived from a processing operation on file f_1. The system shall make sure that f_1' inherits all the jurisdiction restrictions from f_1;

IV. System model

We consider a public $\mathcal{IP}$ managed by a CSP where multiple users lease processing and storage capacity. To benefit from properties such as increased availability, scale flexibility and use of distributed data processing algorithms, data is stored in distributed data stores. This is a common system model in modern infrastructures (including infrastructure clouds) that deal with massive amounts of data and allow data to be reliably stored, replicated and retrieved within a very short time. Examples of such systems are Google BigTable [19], Amazon Dynamo [20], Windows Azure [21], etc. Current distributed data stores store redundant replicas (often *eventually synchonized*) of data on different hosts, thus offering scalability and intra-data center resilience to hardware failures. From a geographical point of view, a distributed storage is either deployed within one data center (and hence in one jurisdiction), or spans several data centres (and possibly several jurisdictions). In the latter case, in order to separate data that is subject to conflicting regulations, users may choose to store the data in two or more distinct IaaS platforms $\mathcal{IP}_1$ and $\mathcal{IP}_2$ – as for example in the case of Amazon Govcloud [22]. However, this restricts geographic redundancy and reduces service availability guarantees. Another possibility is to deploy distributed storage systems across data centres – there are efforts towards this both in academic research ([23]–[25]) and industry implementation[3]. Emerging capabilities of distributed data stores add geographical redundancy, such that the data store is deployed across geographically distinct data centres and offers *inter-data center* redundancy on a global scale while maintaining the *eventual* synchrony of data. For simplicity, we assume a specific subtype of distributed data stores, namely distributed *object storages* depicted in Fig. 1 and described in § IV-A.

According to the model, the domain of the CSP includes the IaaS cloud platform components, the hypervisor, as well as the underlying hardware.

A. Distributed object storage

We assume an object storage capable of storing binary objects (similar to Amazon S3) which can be geographically distributed across multiple data centres[4]. Below, we provide a simplified description of some important components of a distributed object storage (see Fig. 1).

The **API endpoint** is the public API of the object storage. Upon each request, the proxy server looks up the location of the accounts or objects in *the ring*, routes requests and performs error handling.

The ring is a logical structure implemented a distributed hash table that maps the names of stored entities to their physical location. Rings maintain the mapping using *zones*, *devices*, *partitions* and *replicas*. Partitions have a *replication degree* n and their locations are stored in the mapping maintained by the ring. Data can be isolated using the zones of the ring, where each zone can be a data center, switch, cabinet, server or drive. In this model, we assume that each zone is a geographically distinct data centre.

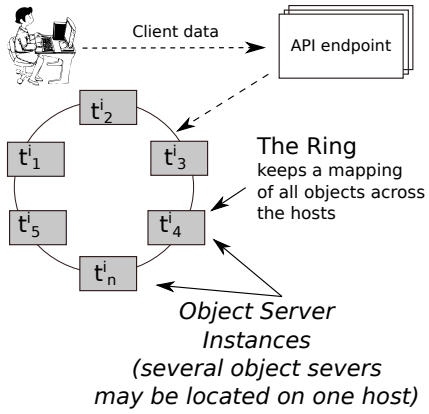

Fig. 1. Sketch of a distributed object storage

Object server is a simple binary large object storage that can *store, retrieve* and *delete* objects on local devices. The object meta data is stored in the *extended file attributes* (otherwise known as "xattrs").

Replicators maintain state consistency in the face of network or node failures. This is done by comparing data with each remote copy to ensure freshness. Replication updates are push-based (rsync file replication to peers).

V. Protocol description

We describe a mechanism used by $\mathcal{IP}$ hosts to acquire their geographical location – mapped to a geolocation cell ($\mathcal{L}$) – and securely store it in TPM PCR$\{15\}$. Next, we present a protocol that allows the data owner to specify the list of geolocation cells where data is stored (or processed) in plaintext (Fig. 5), relying on the TPM functionality to seal the data decryption key to the state of PCR$\{0-7, 15\}$.

A. Geolocation

The proposed solution is based on sealing user data to a set of geolocation cells, making plaintext data only available to hosts physically placed in the jurisdiction authorized by the user. This requires *extending* [26] the platform geolocation cell value $\mathcal{L}$ to a dedicated TPM register during platform boot (we use PCR$\{15\}$). The geolocation cell value is obtained through *reverse geocoding*, which is the process of matching a location point with an administrative unit, e.g. country, region, municipality, etc. For a detailed review of the concept, see [27] [5].

[3] See for example issue HDSF-1432, discussing implementation of HDSF deployment across data centres, *https://issues.apache.org/jira/browse/HDFS-1432*

[4] While this is a fairly recent development, such distributed object storages already exist; see for example https://github.com/openstack/swift/blob/master/CHANGELOG

[5] A sample call to the OpenSteet Map API – http://nominatim.openstreetmap.org/reverse?format=xml&lat=59.406318&lon=17.947&zoom=12&addressdetails=1 – produces the reply presented in Fig. 2

```
<reversegeocode timestamp="Thu, 22 May
    14 08:17:17 +0000">
 <addressparts>
  <suburb>Kista</suburb>
  <city>Stockholm</city>
  <county>Stockholms lan</county>
  <state>Stockholms lan</state>
  <country>Sweden</country>
  <country_code>se</country_code>
 </addressparts>
</reversegeocode>
```

Fig. 2. Sample reverse geocoding result

In our example, reverse geocoding maps any given location point l_i to a geolocation cell $\mathcal{L}$. One approach to supply geolocation cell information to the platform is through out-of-band provisioning at an earlier stage and store it in the TPM, as suggested in [5]. However, such out-of-band provisioning can be prone to administrator error or misuse, therefore we propose to report $\mathcal{L}$ to the TPM through an isolated daemon process ($\mathcal{D}$), and introduce two models for such a daemon to obtain $\mathcal{L}$. The first model (presented in Fig. 3) reuses the concept of "geolocation master nodes" described in [12], where certain hosts are equipped with GPS receivers and dedicated antennas, and are physically separated to reduce the effect of antenna failures, radio interference and spoofing. We assume that the master geolocaton node is previously attested and therefore trusted. The geolocation master node identifies its location point l_i and uses a local *geocoding database* to map l_i to a geolocation cell $\mathcal{L}$. At boot time, $\mathcal{D}$ running on the storage hosts obtains $\mathcal{L}$ from the geolocation master and stores it in the TPM register. Note that this approach does not rule out pre-configuring geolocation cell information out-of-band, as suggested in [5], but reduces the amount of administrative operations required on $\mathcal{IP}$ and eliminates a potential attack vector. In the second model, process $\mathcal{D}$ running on the storage host, uses a navigation device natively attached to the host's motherboard to obtain its location point l_i; next, $\mathcal{D}$ uses a local geocoding database to map l_i to a geolocation cell $\mathcal{L}$; finally, it extends $\mathcal{L}$ into the TPM register (see § III-A). While the feasibility of this model depends on the particular data centre architecture, it *complements* the approach described in model 1 without additional assumptions. We have chosen this latter approach for the rest of the paper, given the wider applications it enables for mobile platforms equipped with a hardware root of trust. The functionality of $\mathcal{D}$ is explained below and presented in Algorithm 1.

In the first step of the protocol, the TTP uses the PCR aggregate of a particular trusted host platform TP_1 to create an asymmetric TPM key pair $SealKey$, which is used to `seal` data to TP_1 in geolocation cell $\mathcal{L}_A$. Consequently, TPM only releases the private $SealKey$ if the current PCR values of TP_1 match the ones specified in the PCR aggregate used in the key creation. For performance reasons, a session key $\mathcal{K}$ – encrypted with the $SealKey$ – is used for data encryption and decryption operations.

The runtime state of the platform with respect to its identity TP_1 and location $\mathcal{L}_A$ is represented by PCR$\{0-7\}$ and PCR$\{15\}$ respectively. These PCRs are populated with values at platform boot time. While the platform state of TP_1 is

input : Navigation Device Port
output: Geolocation cell value hash in TPM PCR 15
initialization: enable GPS, configure GPS Port;
call:fork() to daemonize;
while satellites_tracked < 4 **do**
  Command: get GPGGA data from Navigation Device;
end
call:geolocationcell $a \leftarrow$ *ReverseGeocode*(lat, lon);
call:$H_a \leftarrow$ *SHA1(geolocationcell* a*)*;
call:$H_{exp} \leftarrow$ *Rebuild_Chain(*H_a, H_{def}*)*;
call:$H_{cur} \leftarrow$ *TPM_ReadPCR(*15*)*;
if $H_{cur} \neq H_{exp}$ **then**
  call:TPM_Extend (H_a);
end
sleep(time);

Algorithm 1: Location Reporting Daemon

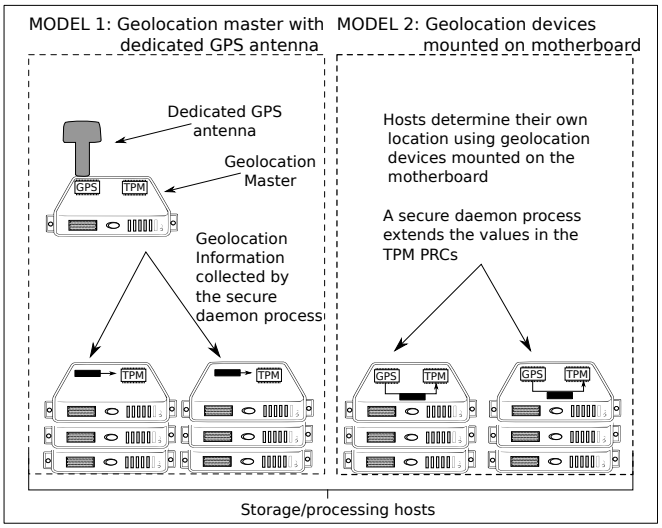

Fig. 3. Geolocation cell reporting to TPM

reported by the TCG-compliant BIOS and the bootloader in PCR$\{0-7\}$, we implement a component $\mathcal{D}$ that runs as a daemon process to report the platform *location* $\mathcal{L}_A$ to PCR$\{15\}$. Component $\mathcal{D}$ is built with minimal functionality: read the navigation device port for location data and extend it to PCR$\{15\}$. The navigation device is enabled at boot time and immediately starts tracking satellites. After a valid location fix is found and the minimum number of required satellites are located[6], $\mathcal{D}$ translates – through reverse geocoding – the location point l_i to the corresponding geolocation cell (i.e. $\mathcal{L}_a$).

Next, the geolocation cell value verification follows. Denote the default value of PCR$\{15\}$ at boot time by H_{def}[7]. $\mathcal{D}$ computes the SHA-1[8] hash (H_a) of $\mathcal{L}_a$ and rebuilds the hash chain for PCR$\{15\}$: H_{def} extended by H_a yields the expected value for PCR$\{15\}$, denoted $H_{exp} = H_{def}||H_a$. Next, $\mathcal{D}$ compares H_{exp} with H_{cur}, the current value in PCR$\{15\}$. If $H_{exp} \neq H_{cur}$, then H_a supersedes H_{cur} and is extended to

[6]A connection to 3 satellites suffices to provide a location point on Earth but a minimum of 4 satellites provides better accuracy.

[7]The value of PCR$\{15\}$ after `TPM_Startup` is 0, as specified in [16].

[8]The choice of SHA-1 is imposed by the TPM v1.2 specifications, see [16]

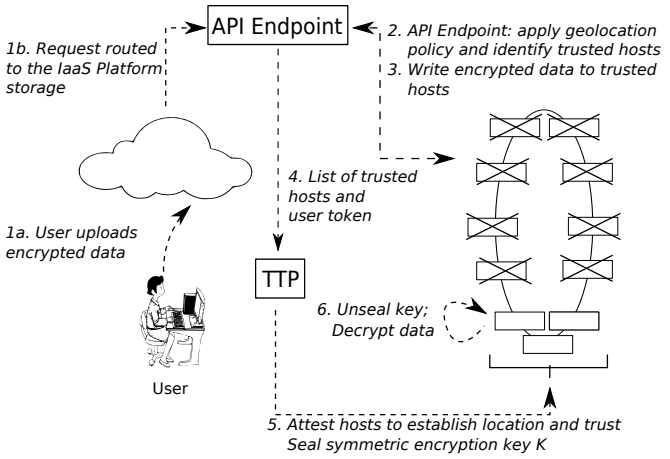

Fig. 4. Process model: data placement with geolocation cell restrictions

PCR$\{15\}$; otherwise, no action is taken. $\mathcal{D}$ runs as a privileged process and performs this operation recurrently. As a result, during the first check PCR$\{15\}$ will be extended with the hash value H_a of the current geolocation cell $\mathcal{L}_a$ and will maintain this value unless platform is relocated to a different geolocation cell, $\mathcal{L}_b$. In that case, $\mathcal{L}_b$ results in a new hash value H_b which supersedes H_{cur} and – according to the procedure above – is extended to PCR$\{15\}$, making $SealKey$ unavailable on the respective platform.

Thus, data encrypted with $SealKey$ is only available in plaintext to TP_1 located in $\mathcal{L}_a$, i.e. preventing access to plaintext data to hosts located outside the authorized jurisdictions. $SealKey$ remains available in the face of restarts, as long as the host is rebooted in the same, trusted state and geolocation cell. Implementation details are provided in §VI.

B. Storage Protection Protocol

We propose the following protocol to ensure that data is available for processing and storage in plaintext only on storage hosts deployed in a user-approved jurisdiction. A high-level model of the protocol is depicted in Fig. 4 and a detailed message flow is presented in Fig. 5. For the purposes of the protocol, we assume that the user u_1 knows the public key of the TTP. We further assume that u_1 can generate a high-entropy symmetric key $\mathcal{K}$ and encrypt own data prior to uploading them to the distributed object storage. Considering the adversary model described in § III-B, we do not explicitly include data integrity protection in the following protocol. However, if integrity of the data is a requirement, the protocol can be easily extended to include it. Finally, we assume that encrypted data is indistinguishable from random noise and encrypted replicas of data may be stored without any legal consequences in any jurisdiction.

Protocol description follows; corresponding steps of the protocol and the message flow are also presented in Fig. 5.

1) User u_1 uploads though the API endpoint the encrypted data (denoted by $E(P, \mathcal{K})$), along with a signed user token containing a description of the location policy constraints – represented by a list of geolocation cells, as shown in Fig. 6 – and the symmetric key $\mathcal{K}$, encrypted with TTPs public key. Thus, $\mathcal{K}$ can be accessed in plaintext only by the TTP (and u_1 who has generated it).
2) The API endpoint applies the provided location policy to determine the set of hosts that are allowed – according to the policy specified by u_1 in the token – to store (or process) the uploaded data in plaintext. Denote this set by $\mathcal{T}_{\mathcal{J}_i}$.
3) The API endpoint writes $E(P, \mathcal{K})$ to the hosts in $\mathcal{T}_{\mathcal{J}_i}$ and returns a write confirmation to the user;
4) The API endpoint forwards to the TTP the encrypted user token and the list in $\mathcal{T}_{\mathcal{J}_i}$.
5) The TTP verifies the authenticity of the user token, performs a remote attestation of the hosts in $\mathcal{T}_{\mathcal{J}_i}$ to verify their software platform state and confirm the claimed location in jurisdiction $\mathcal{J}_i$, decrypts $\mathcal{K}$ and seals it to the trusted configuration of the hosts in $\mathcal{T}_{\mathcal{J}_i}$;
6) The hosts in $\mathcal{T}_{\mathcal{J}_i}$ decrypt $E(P, \mathcal{K})$ and use plaintext data (P) for storage or processing.
7) When requested by user or by the replicator component for synchronization, the storage hosts encrypt data with the same key $\mathcal{K}$ and send $E(P, \mathcal{K})$ to the requester (for clarity, we have omitted this step from figures 4, 5).

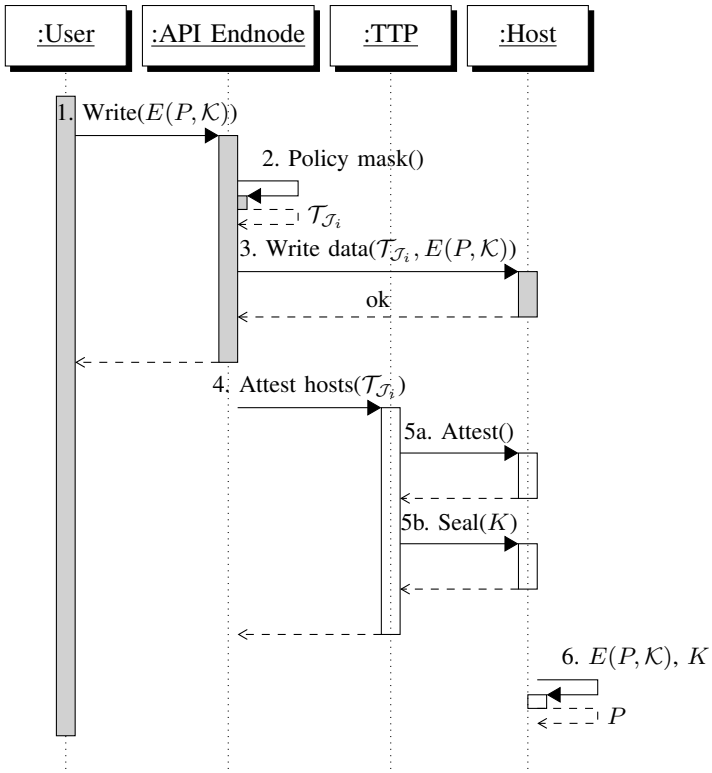

Fig. 5. Protocol message flow: data placement with geolocation cell restrictions.

Note that user u_1 is free to specify either a single or more geolocation cells. The situation when reverse geocoding maps the location point to a wrong geolocation cell is unlikely, and reoccurring errors can be corrected by redefining the boundaries of the respective geolocation cells.

```
<?xml version="1.0" encoding="UTF-8"?>
<LocationPolicy>
    <PolicyID>cred:id</PolicyID>
 <Timestamp>1289123342</Timestamp>
 <GeocellList>
  <Geocell>
     <Location>suburb:name</Location>
     <Location>city:name</Location>
     <Location>county:name</Location>
     <Location>state:name</Location>
  <Location>country:name</Location>
     <Geocell>
 </GeocellList>
</LocationPolicy>
```

Fig. 6. Location policy for a data object

VI. Prototype Implementation

In order to test the validity of the above protocol, we have implemented it using the Swift distributed object storage [9].

Release 1.9.0 of the Swift distributed object storage introduced support for *global clusters*. This includes capabilities for a separate replication network and read/write affinity configuration, which combined enable Swift to run as a single cluster over a wider geographic area, explicitly addressing our assumption about geographically distributed data stores. In addition, Swift provides capabilities to define storage policies based on certain attributes of the storage hosts. For this prototype, we have implemented a policy functionality and configured our Swift deployment to only store the encrypted data on the hosts in a certain jurisdiction $\mathcal{J}_i$. Hosts with the correct location information were able to unseal the encryption key when needed. A modification of the object storage used the unsealed encryption key to decrypt the data before it was written to the node, thus only storing plaintext data on authorized hosts. However, once a file is *requested* from the respective object storage (by the replicator component for container synchronization, or by the API endpoint to be server to the client), it is encrypted with the same encryption key.

We revisit the requirements outlined in § III-C. In protocol context, we identify two classes of objects: *(i)* objects uploaded or created by the data owner; *(ii)* object replicas created by the storage for operational purposes. The objects in class *(i)* are encrypted by the data owner and the decryption key is made available – according to the above protocol – only to TP located in the jurisdiction prescribed by the data owner. Objects in class *(ii)* are *maintained encrypted* and placed on storage hosts at the discretion of the object storage.

For the second requirement, the location policy specified by the object owner enumerates the geolocation cells where objects may be places in plaintext. The TTP will only seal the encryption key on TP that are located in the allowed jurisdiction; the adversary, as defined in the adversary model above, can not force the TTP to seal the encryption key on other hosts.

For the third requirement, note that only storage nodes running on TP have access to the plaintext data for storage and processing. All of the data (f_1') derived through processing will be protected with the same symmetric encryption key prior to being copied by the replicators to other hosts in the ring. In this way, f_1' will inherit the same locations policy: only TP placed in the jurisdiction specified by the owner of object f_1 will have access to the encryption key.

In our implementation, we have used a `Lenovo Thinkpad T430s` host which in a standard configuration has a navigation device `Ericsson H5321 gw Mobile Broadband GPS` attached to the motherboard on a PCIe slot. Navigation devices are widely deployed on mobile platforms (including laptops), and can be easily added to other types of platforms which do not have them pre-installed. Our test host runs Linux (`CentOS 6.4`) which assigns device name `/dev/ttyACM2` to the navigation device according to the default udev rule[9]. In order to have a persistent device name and to ensure that $\mathcal{D}$ reads location data from an authentic source, we define a custom udev rule (shown in listing 7) – protected by including it in the TCB – and place it in `/etc/udev/rules.d/`, such that whenever the navigation device is switched on, a persistent symbolic link `/dev/navigation_device` is created and $\mathcal{D}$ is started.

```
ATTRS{modalias}=="usb:v0B....ip01",
ATTRS{interface}=="Ericsson H5321 GPS",
SYMLINK+="navigation_device",
RUN+=/usr/bin/location_daemon
```

Fig. 7. 10-navigation-device.rules

Given the limitations of GPS signal receivers, placing such receivers on individual platforms might not be a viable solution in many cases, such as data centres located underground or placed in sealed metal containers, as described in [8]. As an alternative, we apply the solution described in [12] and presented in Fig. 3, where a geolocation master server with a dedicated GPS antenna provides signed geolocation cell information to all other hosts in the cluster (e.g. a rack). In this case, the protocol would need to also include the attestation of this dedicated geolocation master. Due to space limitations, we omit the description of these alternative, yet justified scenarios.

For reverse geocoding, we have used the OpenStreet Maps dataset and the API provided by the OpenStreet Maps project[10]. While the size of the complete planet data set is significant (in the range of 400 G), a filtered version containing only country information, produced using a tool such as Osmfilter[11], is significantly smaller – 187 kb – and can be placed on each $\mathcal{IP}$ host and included in the TCB.

VII. Security Analysis

GPS signal security is an important aspect for the presented solution and is an active research topic [28]–[30]. Forays into GPS signal security are out of the scope of this paper and for the purposes of the solution we assume that the setup is capable to detect GPS spoofing attempts, as described in [28], [30]. However, we consider GPS security at application level, to ensure that daemon $\mathcal{D}$ is not tampered and it reads the location data from an interface which is connected to a valid, physical

[9] https://www.kernel.org/pub/linux/utils/kernel/hotplug/udev/udev.html
[10] http://planet.openstreetmap.org/, http://nominatim.openstreetmap.org/
[11] http://wiki.openstreetmap.org/wiki/Osmfilter

navigation device. The integrity of $\mathcal{D}$ is protected by including it in the TCB; moreover, in order to securely and persistently bind the navigation device to a navigation interface, we create a custom `udev rule`, which is also protected by including it in the TCB. Udev rules are loaded by the Linux kernel during the first boot stages and are later used by the udev daemon (i.e. `udevd` which is also a part of TCB) to assign a device name (e.g. `/dev/navigation_device`) when a `uevent` is triggered by that device (e.g. device attached, removed, enabled, disabled, etc.). Including $\mathcal{D}$, `udevd` service and `udev` rules into the TCB prevents their modification under the adversarial model described in (§ III-B).

When it comes to storage and processing of data in the cloud storage, in § V we propose a protocol to both store and process data in plaintext only on trusted hosts in a certain jurisdiction chosen by the user. In both cases, we rely on the remote attestation of hosts (the procedure is described in [18]), expanded with information about the geographical position of the host. The result is, that a host whose platform state has changed through either software modification or change of physical location will not be able to obtain the plaintext version of user data. An adversary in our model might have two goals – obtain the plaintext version of user data or process data in plaintext in a jurisdiction that is not acceptable by the client, in order to reduce operational costs. In both cases, given the physical security of the hosts, confidentiality protection keys will not be made available in case of either a change of the boot aggregate (PCR $\{0-7\}$) or the geographical position of the host PCR$\{15\}$. Here it is worth mentioning that host physical security and change in the geographical position of the platform are not mutually exclusive – considering the example of the modular data centres above, a sealed container with intact platforms may be transferred to a different jurisdiction without affecting the integrity of the platforms.

VIII. Performance

In our experiments, the average time for a commodity GPS receiver from a "cold" start to acquire at least 4 satellites was 97.5 seconds (the GPS device was reset between measurements using the command `AT*E2RESET`). The GPS device maintained a "hot" start between reboots and thus could acquire at least 4 satellites immediately after initialization[12]. Important to note, according to the protocol, acquiring satellites is only necessary at platform boot and does not affect subsequent data access time, we thus exclude this factor from the following performance assessment.

Given the steps of the protocol and considering that the TPM is a relatively slow device, it is to be expected that the largest contributors to the performance impact are the TPM unseal operation and most importantly the data encryption and decryption operations. The TPM unseal operation to obtain the symmetric encryption key, is also a one-time operation, performed once when the respective storage hosts process the data. Once the decryption key is unsealed, it is maintained in memory and used for decryption and encryption of the respective data. In our experiments, the average duration of the unseal operation was 1.23 s. To evaluate the performance of the protocol, we have measured the execution time of a `PUT` operation using file sizes between 1 and 100 M.

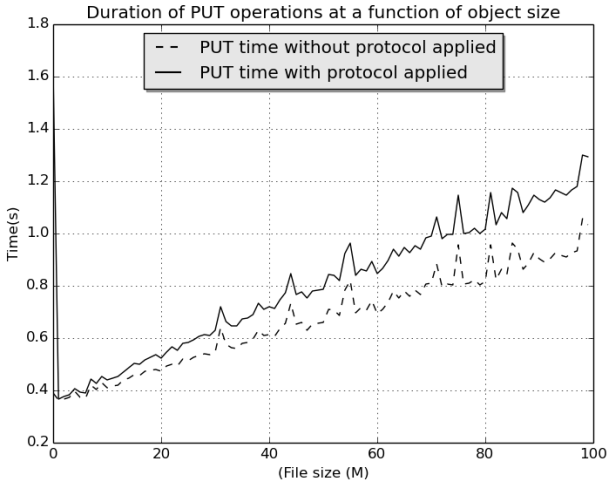

Fig. 8. Performance measurements for PUT operations to Swift distributed object store, with and without the data placement protocol

The results – also presented in Fig. 8 – showed that encrypting objects prior to writing to the object storage implies a relatively small overhead ($\sim$25%) which increases linearly with the size of the file. The initial spike in the graph is due to the relatively long time to unseal the $SealKey$. As it is clearly visible, this does not affect the following PUT operations. The above results may vary, depending primarily on the configured number of replicas and consistency policy. The optimal configuration is use case and deployment architecture specific, and is thus out of the scope of this paper.

IX. Conclusion

In this paper we propose a solution to control the geographic location of plaintext data placed and processed in Infrastructure-as-a-Service deployments. Our analysis of the related work reveals the need to advance beyond verifying that certain data is placed in a certain location. We address this issue and propose several solutions on device, operating system, and object storage platform levels to ensure that data is *only* stored and processed in plaintext in the jurisdiction designated by the data owner. We use trusted computing protocols in order to perform remote attestation of storage and processing hosts, as well as to seal cryptographic material to trusted platforms of the hosts *and their geolocation*, which we obtain from either a commercial off the shelf device on the host motherboard or a dedicated geolocation master node. On the IaaS level, we leverage the trusted state of the platform to decrypt, store and process user data in plaintext only on hosts located in a certain jurisdiction specified by the user prior to upload. Our performance tests have demonstrated the feasibility of the proposed approach; further improvements in CPU architectures are expected to reduce the overhead induced by the data encryption stages of the algorithm. Future work includes refinement of the proposed geolocation cell model, integration of platform attestation with kernel-level mandatory

[12] *Cold Start*: The receiver must download almanac and ephemeris information to achieve a position fix. *Hot Start*: A hot start occurs when a receiver has up-to-date almanac and ephemeris information. http://www.ni.com/white-paper/7189/en/

access control policies as well as minimization and eventual elimination of the TTP, which would allow us to consider stronger adversary models.

REFERENCES

[1] D. Dhar and J. H.-e. Ho, "Stem cell issue: stem cell research policies around the world," The Yale journal of biology and medicine, vol. 82, no. 3, p. 113, 2009.

[2] A. Albeshri, C. Boyd, and J. G. Nieto, "GeoProof: Proofs of Geographic Location for Cloud Computing Environment," in Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on. IEEE, 2012, pp. 506–514.

[3] A. Juels and A. Oprea, "New Approaches to Security and Availability for Cloud Data," Commun. ACM, vol. 56, no. 2, pp. 64–73, Feb. 2013. [Online]. Available: http://doi.acm.org/10.1145/2408776.2408793

[4] D. Desai, "Law and Technology Beyond Location: Data Security in the 21st Century," Communications of the ACM, vol. 56, no. 1, pp. 34–36, 2013.

[5] E. Banks, M. Bartock, K. Firftal, D. Lemon, K. Scarfone, U. Shetty, M. Souppaya, T. Williams, and R. Yeluri, "DRAFT Trusted Geolocation in the Cloud: Proof of Concept Implementation," NIST special publication, vol. 7904, p. 42, 2012.

[6] Z. N. Peterson, M. Gondree, and R. Beverly, "A position paper on data sovereignty: The importance of geolocating data in the cloud," in Proceedings of the 8th USENIX conference on Networked systems design and implementation, 2011.

[7] M. Gondree and Z. N. Peterson, "Geolocation of data in the cloud," in Proceedings of the third ACM conference on Data and application security and privacy. ACM, 2013, pp. 25–36.

[8] J. Hamilton, "Architecture for modular data centers," arXiv preprint cs/0612110, 2006.

[9] S. Toor, R. Toebbicke, M. Z. Resines, and S. Holmgren, "Investigating an open source cloud storage infrastructure for CERN-specific data analysis," in Networking, Architecture and Storage (NAS), 2012 IEEE 7th International Conference on. IEEE, 2012, pp. 84–88.

[10] C. Krauß and V. Fusenig, "Using trusted platform modules for location assurance in cloud networking," in Network and System Security. Springer, 2013, pp. 109–121.

[11] N. Paladi, C. Gehrmann, and F. Morenius, "Domain-Based Storage Protection (DBSP) in Public Infrastructure Clouds," in Secure IT Systems. Springer, 2013, pp. 279–296.

[12] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild et al., "Spanner: Googles globally-distributed database," in Proceedings of OSDI, vol. 1, 2012.

[13] P. Mell and T. Grance, "The NIST definition of cloud computing (draft)," NIST special publication, vol. 800, 2011.

[14] J. Pearsall and P. Hanks, The new Oxford dictionary of English. Clarendon Press, 1998.

[15] C. Gentry, "Computing arbitrary functions of encrypted data," Communications of the ACM, vol. 53, no. 3, pp. 97–105, 2010.

[16] Trusted Computing Group, "TCG Specification, Architecture Overview, revision 1.4," Trusted Computing Group, Tech. Rep., 2007.

[17] N. Santos, K. P. Gummadi, and R. Rodrigues, "Towards Trusted Cloud Computing," in Proceedings of the 2009 Conference on Hot Topics in Cloud Computing, ser. HotCloud'09. Berkeley, CA, USA: USENIX Association, 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855533.1855536

[18] N. Paladi, C. Gehrmann, M. Aslam, and F. Morenius, "Trusted launch of virtual machine instances in public IaaS environments," in Information Security and Cryptology–ICISC 2012. Springer, 2013, pp. 309–323.

[19] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," ACM Transactions on Computer Systems (TOCS), vol. 26, no. 2, p. 4, 2008.

[20] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in ACM SIGOPS Operating Systems Review, vol. 41, no. 6. ACM, 2007, pp. 205–220.

[21] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci et al., "Windows azure storage: a highly available cloud storage service with strong consistency," in Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. ACM, 2011, pp. 143–157.

[22] Amazon, "Amazon Web Services," http://aws.amazon.com/es/ec2/, [Online; accessed 20-Apr-2014].

[23] Y. Sovran, R. Power, M. K. Aguilera, and J. Li, "Transactional storage for geo-replicated systems," in Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. ACM, 2011, pp. 385–400.

[24] C. Li, D. Porto, A. Clement, J. Gehrke, N. Preguiça, and R. Rodrigues, "Making geo-replicated systems fast as possible, consistent when necessary," in Proceedings USENIX Symposium on Operating System Design and Implementation (OSDI), 2012.

[25] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen, "Stronger semantics for low-latency geo-replicated storage," in Symposium on Networked Systems Design and Implementation, 2013.

[26] "TPM Specification, TPM Main Part-III Design Principles," http://www.trustedcomputinggroup.org/resources, July 2007, [Online; accessed 20-Apr-2014].

[27] D. W. Goldberg, J. P. Wilson, and C. A. Knoblock, "From text to geographic coordinates: the current state of geocoding," URISA journal, vol. 19, no. 1, pp. 33–46, 2007.

[28] C. J. Wullems, "A Spoofing Detection Method for Civilian L1 GPS and the E1-B Galileo Safety of Life Service," Aerospace and Electronic Systems, IEEE Transactions on, vol. 48, no. 4, pp. 2849–2864, 2012.

[29] B. W. O'Hanlon, M. L. Psiaki, J. A. Bhatti, D. P. Shepard, and T. E. Humphreys, "Real-Time GPS Spoofing Detection via Correlation of Encrypted Signals," Navigation, 2013.

[30] M. L. Psiaki, B. W. O'Hanlon, J. A. Bhatti, D. P. Shepard, and T. E. Humphreys, "Gps spoofing detection via dual-receiver correlation of military signals," Aerospace and Electronic Systems, IEEE Transactions on, vol. 49, no. 4, pp. 2250–2267, 2013.